Doom Reinforcement Learning - Report

Adam Kaniasty, Hubert Kowalski, Norbert Frydrysiak, Igor Kołodziej, Krzysztof Sawicki

July 11, 2024

Abstract

This report describes the implementation and performance evaluation of a model designed to complete the **ViZDoom**[2] scenarios. The agent training process utilized two machine learning methods: Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C). Both methods are based on reinforcement learning, where the agent learns through interaction with the environment and receiving rewards for performed actions. The report includes a description of the model implementation and the results of the conducted studies, which involved selecting appropriate reward function parameters and training duration to ensure the agent completes the level most efficiently. The studies included a direct performance comparison of the A2C and PPO models, which showed that the PPO method achieved better results.

Contents

1 Introduction

This report presents the results of the project for the ”Research Workshops” course. Our goal was to solve the game Doom using methods from the field of Reinforcement Learning. For this purpose, we created two models: (A2C and modified PPO) and designed two reward functions. During the research, we conducted performance tests to select the best set.

1.1 Algorithm Concept Description

Reinforcement learning (RL) [5] is a machine learning method in which an agent learns by interacting with the environment and receiving rewards for performed actions. The main elements of RL are:

- Agent: A program or algorithm that takes actions.
- Environment: Everything the agent interacts with.
- Actions: Possible decisions or moves the agent can make.
- State: The current situation or configuration of the environment.
- Rewards [1]: Feedback from the environment indicating how good the agent’s action was.
- Policy: A strategy that defines what action the agent should take in a given state.

The RL agent focuses on maximizing the sum of rewards it receives over the entire duration of the game. The learning process involves modifying the policy based on observed progress.

1.2 ViZDoom

ViZDoom is a research tool and platform used for training and testing artificial intelligence algorithms, particularly in the context of Reinforcement Learning. ViZDoom allows AI agents to play the game Doom by providing a 3D environment reflecting the game state. The gameplay is based on a provided game engine that enables, among other things, three-dimensional movement. ViZDoom has a Python API, making it easy to integrate with machine learning tools and libraries such as TensorFlow and PyTorch.

[1]Rewards can be positive (for good decisions) or negative (for bad decisions)

1.3 Game Description

"Doom" is a classic first-person shooter (FPS) computer game created by id Software and first released on December 10, 1993. It is one of the most influential games in the history of the computer game industry, having had a huge impact on the development of the FPS genre. In our project, our main scenario was the "Death Corridor" level[2], which is often used as a benchmark in reinforcement learning research. It is characterized by:

- A narrow corridor - The player moves through a corridor, preventing them from avoiding enemies.
- A large number of enemies - Their number can be adjusted, affecting the difficulty level.
- Limited resources - The player has a finite amount of ammunition and health.

The goal of the level is to reach the end of the corridor. The difficulty level can be adjusted using a parameter. We also tested our model on the "Basic" and "Defend Center" scenarios. The first one involved hitting an enemy located directly in front of the agent. In the second scenario, enemies approached the agent, and the agent had to turn and kill them before they reached him.

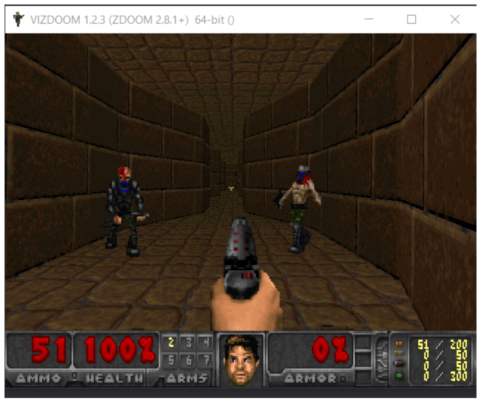

Figure 1: Screenshot of Death Corridor gameplay.

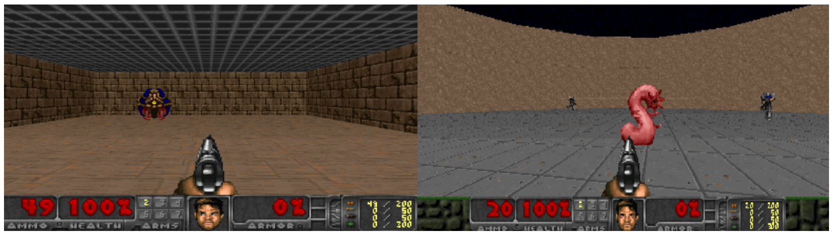

Figure 2: Screenshot of Basic and Defend Center gameplay.

2 Algorithms

The PPO (Proximal Policy Optimization) [1] and A2C (Advantage Actor-Critic) [4] algorithms focus primarily on the neural network architecture used to represent the policy network and value function [3].

1. **Policy (Actor):** A neural network that takes the state of the environment as input and outputs a probability distribution of actions. This distribution represents the agent's policy, determining the probabilities of taking each action in a given state.
2. **Value Function (Critic):** A neural network that takes the state as input and predicts the value of that state, i.e., the expected cumulative reward.

[2]This level was created by the player community, not by the main publisher

These algorithms work similarly, but PPO enhances the policy update concept of A2C and applies clipping [6] to ensure training stability.

2.1 Algorithm Operation

1. **Initialization:** We initialize two neural networks: one for the policy (actor) and one for the value function (critic).
2. **Environment Interaction:** The agent observes the current state s_t. The policy (actor) selects an action a_t based on the probability distribution output from the current state s_t.
3. **Action Execution:** The action a_t is executed in the environment, leading to a new state s_{t+1} and a reward r_{t+1}.
4. **Value Calculation:** The value function (critic) evaluates the state s_t and predicts its value $V(s_t)$. The value function also evaluates the new state s_{t+1} and predicts its value $V(s_{t+1})$.
5. **Advantage Calculation:** We calculate the advantage $A(s_t, a_t)$, which is the difference between the received reward plus the future state's value and the current state's value:

   $$A(s_t, a_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

   where γ is the discount factor.
6. **Network Update:**
   - **Policy Update (A2C):** The policy gradient is updated based on the advantage:

     $$\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

     where π_θ is the policy parameterized by θ.
   - **Policy Update (PPO):** The policy is updated with a constraint using clipping [1]:

     $$L^{CLIP}(\theta) = E_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

     where:
     - $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ - the ratio of new and old probabilities,
     - ϵ - the parameter determining the degree of clipping.
   - **Value Function Update:** The value function is updated based on the mean squared error (MSE):

     $$\text{MSE} = (r_t + \gamma V(s_{t+1}) - V(s_t))^2$$
7. **Repeat Steps:** Steps 2-6 are repeated for a specified number of epochs or training steps.

2.2 Neural Network Parameters

During our research, the state space consisted of sequential data (game data) and structural data (game images) [4]. Therefore, we used a CNN (Convolutional Neural Network) adapted to this, with convolutional layers. The ReLU activation function was used. The output consisted of a linear layer. To improve agent training efficiency, we introduced image preprocessing by removing unnecessary game state information. The image was converted to grayscale and rescaled. The status bar was cropped to eliminate irrelevant data. Additionally, we generated an image version containing only edges to highlight important scene elements.

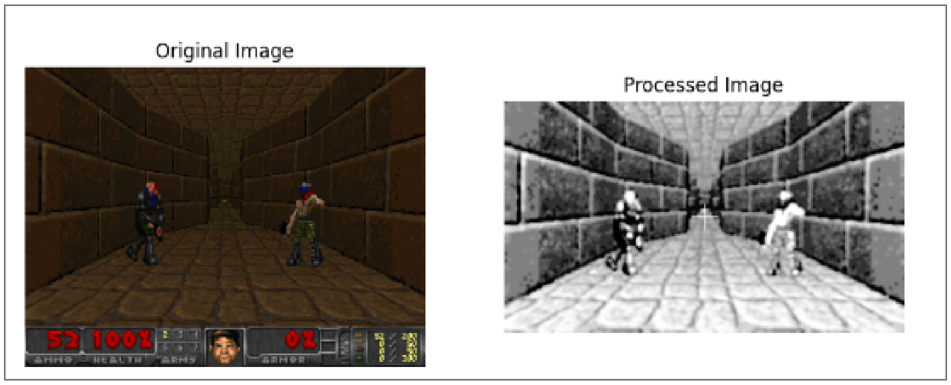

Figure 3: Game image preprocessing.

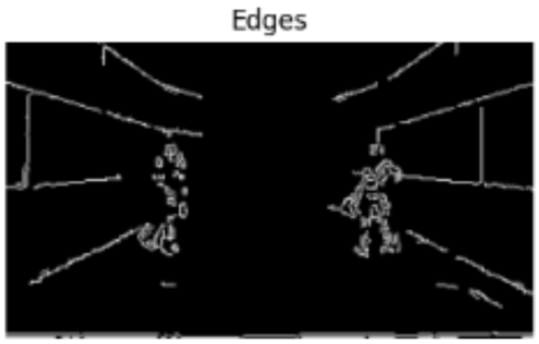

Figure 4: Game image with highlighted edges.

2.3 Hyperparameters

During the training process, we identified four hyperparameters that significantly influenced the course and speed of training. They are (chosen values in parentheses):

- N Steps (4096 or 8192) - time interval regulating network updates.
- Gamma (0.95) - discount factor for rewards; increasing it encourages the agent to take actions with long-term benefits.
- Clip range (0.01) - prevents very large changes in policy.
- Entropy Regularization (0.05) - introduces a penalty for actions the agent is very certain about. This prevents premature deterministic behaviors.

3 Reward Function

The implemented reward function took several factors into account. The model's goal was to collect the armor located at the end of the corridor. Before that, it should kill the enemies in the corridor to avoid losing too much health. Addressed game state changes:

- Health - the agent was penalized for losing health.
- Ammunition - the agent was penalized for using too much ammunition to discourage constant shooting.
- Kills - the agent was rewarded for each kill to encourage fighting enemies.
- X change - the agent received rewards for moving forward. However, in our reward function, we introduced a penalty for this action to balance the built-in reward system and allow for scaling all rewards. We added our reward to the default reward. Usually, the agent received a net positive reward for moving forward, while moving backward resulted in a negative reward equal in magnitude to the forward reward. The reward was calculated based on the change in the X-axis coordinates. In further chapters, we will provide the net reward for this action. If it is positive, it means we reward forward movement and penalize backward movement.

Additionally, the agent received a reward for completing the level by collecting the armor. The specific reward values were experimentally selected based on training observations.

4 Metrics

To verify the correctness of the training process, we introduced metrics to monitor progress. Combined with gameplay observation, they provided a comprehensive view of the model's capabilities. The introduced metrics are as follows:

- Episode Ammo - measures the amount of ammunition used in each episode.
- Episode Distance - measures the distance traveled from point (0,0,0) in 3D space in each episode.
- Episode Health - measures the health at the end of the episode. The metric returns a non-zero value in two cases:
  - the agent reached the end of the corridor and collected the armor
  - the episode's allotted time expired
  Otherwise, the agent died, and its health was 0.
- Episode Killcount - counts the number of enemies killed during one episode. The goal is to achieve a value of 6, as there are that many enemies in the entire corridor.
- Episode Return - measures the return received by the agent in each episode. Its value is lowered by the gamma parameter.
- Episode Steps - measures the number of steps taken in each episode.
- Timestep Reward - measures rewards in each timestep. Rewards are added to the total return and scaled by the gamma factor.

5 Training Process

5.1 Initial Training

In the initial training phases, the PPO model was used and trained for 100k timesteps at difficulty level 2. Only the custom reward was considered, i.e., the built-in environment reward was not added to the total. The reward parameters were as follows:

Parameter	Value
Lost health point	-11
Enemy kill	210
Used ammunition	-2
X change	+1
Death	-100

Table 1: Table showing the reward parameters used in the initial training phases of the PPO model in the Death Corridor scenario.

During training, the policy was non-deterministic, so the agent sometimes chose a random move, whereas during testing, we mainly looked at the deterministic policy, i.e., the agent always chose the moves it had learned were the best. As a result, during the training process, the number of episodes played was 549, of which the agent died 545 times, and the remaining 4 episodes ended with the time allotted to complete the level expiring. The agent killed an average of 0.53 enemies per episode. During testing, the high reward for moving forward caused a tendency to move forward without paying attention to enemies. Small damages dealt by enemies made the agent reach the end of the corridor, reinforcing its strategy's correctness. The metrics also did not indicate correct training.

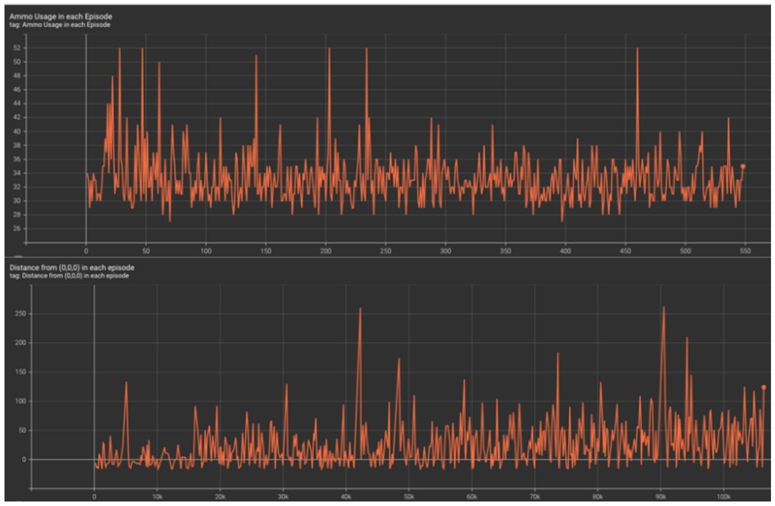

Figure 5: Plot of ammo usage metric value per episode and distance from (0,0,0) metric value per timestep.

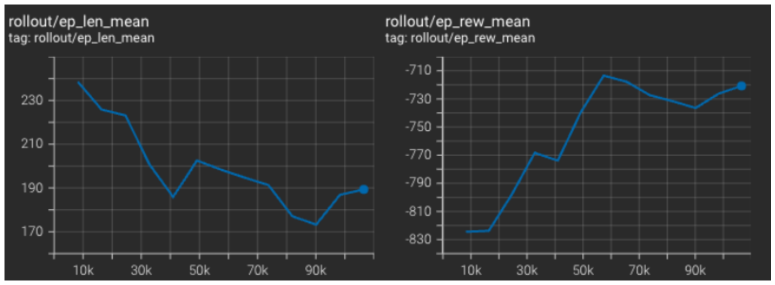

Figure 6: Plot of episode length and episode mean reward metrics value per training iteration.

The next training phases involved changing individual values in the reward function and longer agent training to encourage fighting enemies. As a result of this adjustment, the agent began to overuse the shooting action associated with killing an enemy. Simultaneously, it stopped moving forward and focused only on the first two enemies.

5.2 Final Results

The solution that produced the desired effect was changing the agent's input during training. Instead of state information in the form of scalar data and images, it received only images. This reduced the learning complexity and made it easier for the agent to recognize enemies. This approach was tested in all three scenarios.

5.2.1 Basic

The chosen PPO model was trained for 100k timesteps.

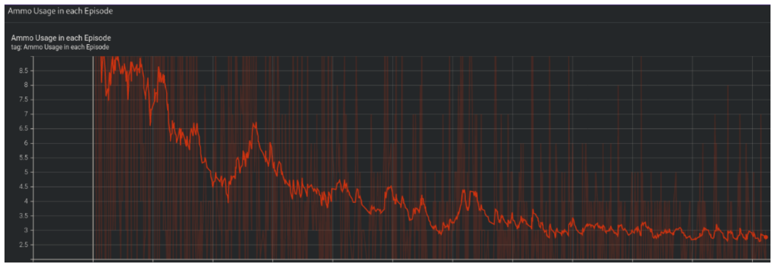

Figure 7: Plot of ammo usage metric value per episode. The downward trend suggests the agent needed fewer bullets to kill the enemy.

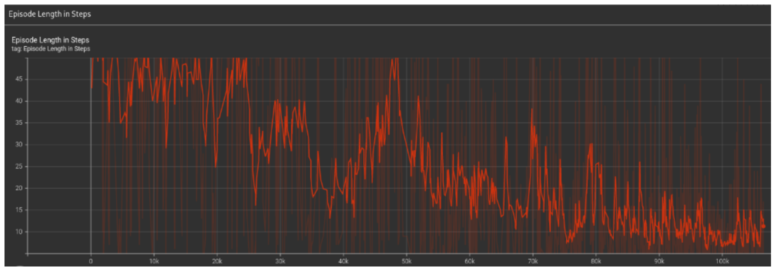

Figure 8: Plot of episode length metric value per timestep. The decrease in episode time suggests the agent killed the enemy faster.

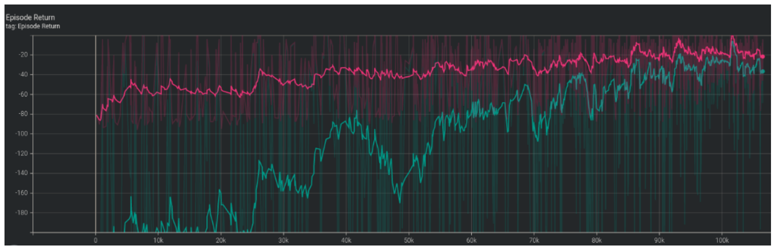

Figure 9: Plot of episode return metric value. The pink line shows the discounted value.

5.2.2 Defend Center

We used the same model, already trained on the Basic scenario. For the defend the center scenario, it was further trained for 200k timesteps. In the presented plots, the blue line represents training during the first 100k timesteps, and the orange line represents the next 100k timesteps. The ammo usage plot shows the amount of ammunition used in each episode. The orange line shows fewer measurements in this metric, indicating that the agent played fewer episodes in the second half of the training, surviving longer.

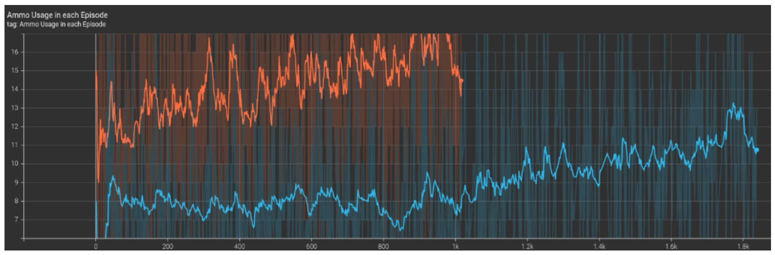

Figure 10: Plot of ammo usage metric value.

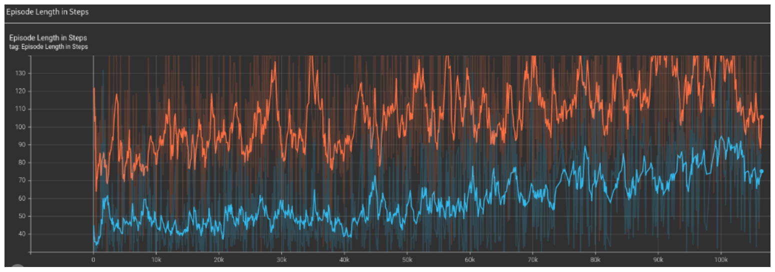

Figure 11: Plot of episode length metric value.

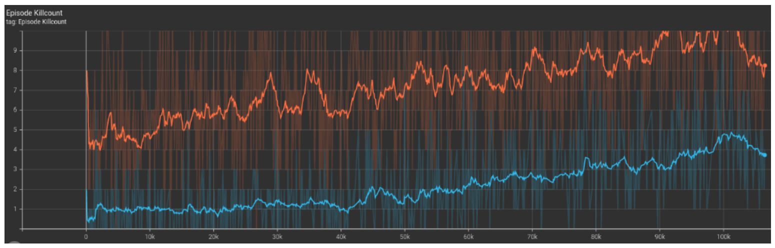

Figure 12: Plot of killcount metric value.

As we can see in the above plots, all metrics increased over time. The increase in ammo usage suggested that the agent shot at more enemies, which is confirmed by the kill count metric.

5.2.3 Death Corridor

In addition to the training process itself, we changed the reward function parameters compared to the initial training.

Parameter	Value
Lost health point	-1
Enemy kill	200
Used ammunition	-5
X change	+1
Death	-200

Table 2: Table showing the reward parameters used during the final models' training.

The PPO model was trained in 5 stages: 100k, 200k, 300k, 400k, 500k. In addition to PPO, A2C was also trained, which performed significantly worse than PPO. As a result, its training was abandoned. The metrics are as follows (legends in all plots mean the same, where 1 means 100k, 2 means 200k, etc.):

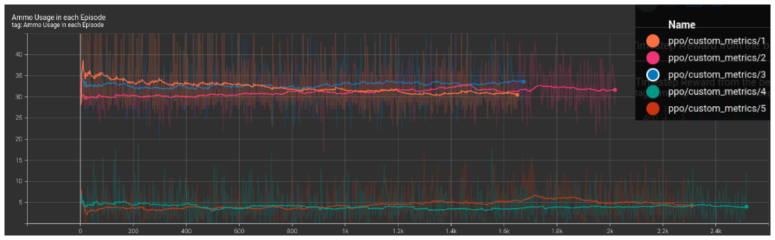

Figure 13: Plot of ammo usage metric value. The length of the plots varies depending on the stage, as each stage lasts 100k steps, and different numbers of episodes can occur during it. In simpler difficulty levels, episodes lasted on average longer.

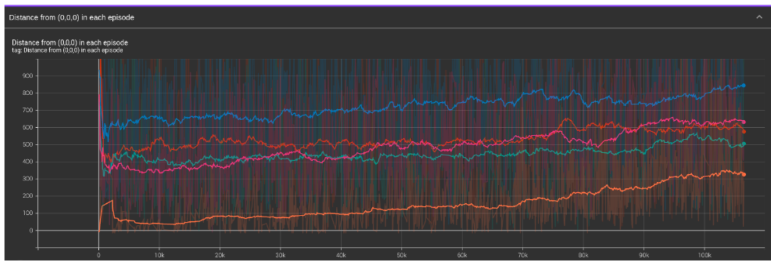

Figure 14: Plot of episode distance metric value.

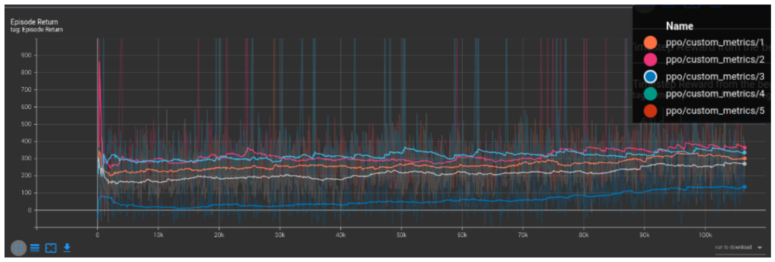

Figure 15: Plot of episode return metric value.

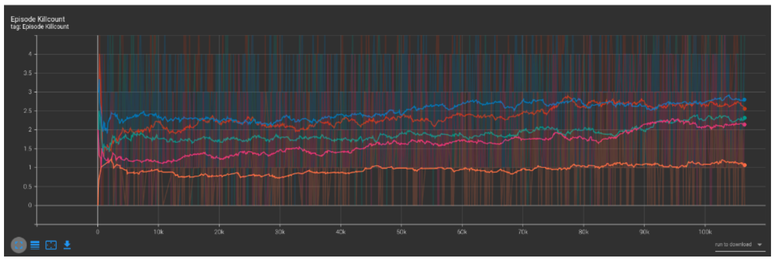

Figure 16: Plot of killcount metric value.

6 Conclusions and Summary

Extending the training process significantly improved the results in almost every analyzed metric. The agent used less ammunition because its accuracy increased, and the reward, kill count, and episode length increased. Based on the experiments, it can be concluded that the PPO model performs better than A2C. The key to achieving correct training was the proper selection of values in the reward function and training with only images using a CNN network. The idea of additional training with scalar data proved to be unsuccessful. At the same time, it is not certain whether edge detection preprocessing was significantly necessary. However, the final results are satisfactory. In the case of the Basic scenario, the agent performed significantly better than a real player. Similarly, in Defend Center, an untrained person would not be able to kill as many enemies. In the case of the "Deadly Corridor" scenario, it can be debated who would perform better. However, unlike a human, the agent has unlimited development potential and no biological barriers affecting reaction speed and decision-making.

7 Further Development Opportunities

Among the available ViZDoom scenarios, many have not yet been tested (deathmatch, maze). There, the model's improvement can be sought. Additionally, a duel between the agent and a human could provide information about the model's real reaction time and accuracy when the opponent is less predictable. Of course, we are not sure if the chosen reward function parameters are the most beneficial. Further development could go towards re-optimization, and in the best case, automating this process similarly to grid search methods.

References

[1] John Schulman i in. *'Proximal Policy Optimization Algorithms'*. URL: `https://arxiv.org/abs/1707.06347`.

[2] Michał Kempka i in. *'ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning'*. URL: `https://arxiv.org/abs/1605.02097`.

[3] Thomas Simonini i in. *'The intuition behind PPO'*. Dostęp: 09.06.2024. URL: `https://huggingface.co/learn/deep-rl-course/unit8/intuition-behind-ppo`.

[4] Volodymyr Mnih i in. *Asynchronous Methods for Deep Reinforcement Learning.* URL: `https://arxiv.org/abs/1602.01783`.

[5] Richard S. Sutton and Andrew G. Barto. *'Reinforcement Learning: An Introduction'*. 2018.